VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Hra Munchkin v prostředí internetu

Game Munchkin over Internet

2014 David Lefnar

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **David Lefnar**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Hra Munchkin v prostředí internetu
Game Munchkin over Internet

Zásady pro vypracování:

Cílem práce je vytvořit program pro hraní hry Munchkin v prostředí internet. Program bude naimplementován v jazyce Java, bude se skládat ze serverové části, která bude umožňovat připojování jednotlivých uživatelů. Klientská část pak bude samostatná aplikace využívající technologie Java WebStart, která se připojí k serveru a umožní uživateli hrát hru s ostatními hráči.

Systém umožní:
1. Hraní hry více hráčů v prostředí internetu.
2. Využití hráče s jednoduchou umělou inteligencí.
3. Textovou a hlasovou komunikaci hráčů.
4. Hraní hry dle pravidel (boj, útěk, změna rasy, změna pohlaví, zbrojení, používání karet).
5. Algoritmizace významu karet pomocí rozhraní.
6. Interakce hráčů v rámci jednoho tahu (pomoc, znesnadnění situace, vyjednávání).
7. Program umožní připojení do současně vyvíjeného portálů her v jazyce Java.

Práce bude obsahovat:
1. Popis použitých technologií.
2. Implementaci výše popsané hry.
3. Programátorskou dokumentaci řešení s využitím diagramů jazyka UML.

Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
[2] DARWIN, Ian F. Java cookbook. 2nd ed. Sebastopol, CA: O'Reilly, c2004, xxiv, 829 p. ISBN 05-960-0701-9. Dostupné z: http://it-ebooks.info/book/2249/

Dále podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny, publikace a ostatní zdroje, ze kterých jsem čerpal.

V Ostravě 30. dubna 2014

.....

Rád bych poděkoval Ing. Davidovi Ježekovi, Ph.D. za cenné rady, připomínky a vstřícnost projevenou při konzultacích a vypracování bakalářské práce.

Abstrakt

Bakalářská práce popisuje technologie a praktiky, jež jsem využil při vývoji hry Munchkin v prostředí internetu v programovacím jazyce Java. Charakterizuje typické rysy samotné hry a jejich reprezentaci ve zdrojovém kódu. Pro tvorbu uživatelského rozhraní se využívá komponent knihovny swing, manipulace s nimi je také součástí této práce. Vedle oblasti implementace přenosu a zpracování hlasu se zmiňuji také o komunikaci v síti internet nad protokolem TCP a UDP, umělé inteligenci a její roli v tomto projektu.

Klíčová slova: Munchkin, hra, Java, síťová komunikace, internet, objektové proudy, hlasový chat, textový chat, swing, umělá inteligence

Abstract

Bachelor thesis describes technologies and practices that I have used in the development of Munchkin game over Internet in Java programming language. It characterizes typical features of the game itslef and their representation in source code. For the creation of the user interface were used components of swing library, their handling is also included in the thesis. In addition to area of voice transfering and processing implementation I also mention communication within the Internet over TCP and UDP protocols, artificial intelligence and its role in this project.

Keywords: Munchkin, game, Java, network communication, internet, object streams, voice chat, text chat, swing, artificial intelligence

Seznam použitých zkratek a symbolů

AWT	–	Abstract Windowing Toolkit
GUI	–	Graphical User Interface
IBM	–	International Business Machines Corporation
IP	–	Internet Protocol
I/O	–	Input/Output
JAR	–	Java Archive
JFC	–	Java Foundation Classes
JNI	–	Java Native Interface
JNLP	–	Java Network Launch Protocol
JVM	–	Java Virtual Machine
LIFO	–	Last In First Out
MVC	–	Model-View-Controller
P2P	–	Peer-to-Peer
SWT	–	Standard Widget Toolkit
TCP	–	Transmission Control Protocol
UDP	–	User Datagram Protocol
UML	–	Unified Modeling Language
URL	–	Uniform Resource Locator

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Tématem bakalářské práce je vývoj hry Munchkin v prostředí internetu. Cílem je vytvořit komplexní, plnohodnotnou aplikaci umožňující komunikaci více klientů mezi sebou v programovacím jazyce Java. V tom je obsaženo i zvládnutí analýzy karet a jejich významů pro pozdější implementaci a s tím spjatá herní logika, která je založena zejména na vzájemné interakci hráčů. Ve hře budou zahrnuty typické akce charakterizující hru Munchkin jako žádání o pomoc, znesnadňování postavení ve hře ostatními hráči a akce plynoucí z významu jednotlivých karet.

Dále se budu zaměřovat na komunikaci vzdálených bodů v prostředí internetu nad protokoly umožňující přenos objektů plynoucích z analýzy. Jedním z objektů s jistotou bude hlas, protože dalším cílem je do klientské části začlenit hlasový chat, aby se mohli hráči mezi sebou rychleji a pohodlněji domlouvat, dohadovat a vyjednávat nad herními postupy. Hráči budou mít více možností dorozumívání. Hráči, kterým hlasový chat nebude vyhovovat, nebo ho z nějakého jiného důvodu nebudou využívat, mají na výběr možnost použít pouze klasický, textový chat.

Dalším cílem je integrace do právě portálu her vyvíjeného jedním z mých kolegů a s tím (částečně) spojená implementace jednoduché umělé inteligence. Umělá inteligence je začleněna hlavně z důvodu správného návrhu a rozložení softwaru pomocí logického rozdělení, ale také, aby byla schopna samostatně fungovat a „nahradit“ člověka jako hráče této hry. Zmiňovaný herní portál bude, mimo jiné, udržovat tuto hru ve své správě a bude tvořit prostředníka mezi hráči a hrou, tak, jako to známe například z populárního herního portálu GamePark.

Jedním ze specifických a osobních cílů je zrealizovat tento projekt nezávisle na zkonkretizovaných nástrojích a frameworcích, a k práci využít pouze obecné praktiky a základní stavebními prvky platformy Java.

Přestože tato práce neplní funkci manuálu hry Munchkin, jsou ve velice stručné podobě v první kapitole zmíněna pravidla hry. Je to z důvodu nabytí lepšího povědomí o problémech, se kterými jsem se musel vypořádat, a zasazuje to zbytek práce do kontextu. Pokračuje se popisem základních i pokročilých programátorských praktik využitých při vývoji, analýzou a algoritmizací karet a jejich významů. Dále se zmiňuji o možnostech tvorby uživatelského rozhraní, kde blíže popisuji mnou vybranou alternativu. Asi nejrozsáhlejší kapitola se týká komunikace a technologií využitých při její implementaci. Předposlední kapitola obeznamuje čtenáře s integrací do výše zmiňovaného herního portálu. Práci zakončuje kapitola týkající se umělé inteligence.

2 Munchkin

Munchkin je dobrodružná karetní hra pro tři až šest hráčů, představující parodii na všechny RPG hry (role-playing game – hra na hrdiny [1]). Každý z účastníků se snaží zabíjet nestvůry, sbírat poklady a znemožňovat vyhrát ostatním protihráčům na cestě za dosahováním vyšších úrovní až k samotnému vítězství. Parodie a satira se dotýká celé hry, počínaje zpracováním pravidel, skrze vtipné a často zesměšňující obrázky a popisky jednotlivých kartiček, až po možné akce samotné.

Popisovaná a originální podoba hry pochází z dílny Seve Jackson Games, kde ji navrhl sám Američan Steve Jackson roku 2001. Veškerou ilustraci má na svědomí John Kovalcic [2]. Vedle této verze hry obsahující 168 kartiček, je dostupný i nespočet dalších rozšiřujících verzí [2].

2.1 Pravidla

Jednotliví hráči jsou ve hře zastoupeni hrdiny. Hrdinové jsou charakterizování rasou, povoláním, pohlavím, úrovní a silou. Na začátku hry jsou všichni hrdinové úrovně jedna, rasy člověk, bez povolání a druh pohlaví hrdiny kopíruje pohlaví hráče. Síla se vypočítává jako součet úrovně hrdiny a bonusů karet k tomu určených. Úrovně je možno získat zabitím nestvůr, nebo důsledkem efektu karet. Vyhrát lze jediným způsobem, dosažením hrdiny desáté úrovně.

Před samotným začátkem hry se rozdělí kartičky do dvou balíčků na poklady a dveře. Do kterého balíčku karta patří, je znázorněno na její zadní straně obrázkem. Každý z hráčů dostane do ruky 4 karty z obou balíčků. Hraní se provádí v rámci kola a jeho fází. Před první kolem může hráč karty, u kterých je to dovoleno, vyložit před sebe a následuje samotné kolo.

Každé kolo začíná možností upravit své karty ve hře a vyložit nové karty z ruky. Pokračuje se fází zvanou „vykopnutí dveří“, která označuje akci položení karty z balíčku dveří do hry, otočenou vzhůru přední stranou. Když je kartou nestvůra, začíná boj. Je-li kartou kletba, je aplikován její efekt. Pokud kartou není ani nestvůra ani kletba, může ji hráč vložit do své ruky, nebo ji ihned zahrát.

V případě, že po vykopnutí dveří nebyla nalezena nestvůra, má hráč možnost hrát nestvůru ze své vlastní ruky. Tato fáze kola se nazývá „hledání obtíží“. Nebojoval-li hráč s monstrem, ať už po vykopnutí dveří nebo při hledání obtíží, vytáhne další kartu z balíčku dveří, ale tentokrát ji umístí rovnou do své ruky, aniž by ji ukazoval ostatním. Než přijde na řadu další hráč, je potřeba zredukovat počet karet v ruce na pět. Toho se dosáhne odehráním dostatečného počtu karet během předchozích fází kola, odevzdáním přebytečných karet hráči s nejnižší úrovní, nebo, má-li hráč samotný nejnižší úroveň ve hře, jsou karty odhozeny ze hry [2].

2.1.1 Boj

Vyhodnocení boje se provádí porovnáním síly hrdiny a nestvůry. Hráč boj vyhrává pouze v situaci, kdy je síla jeho hrdiny ostře větší než síla nestvůry. V takovém případě nestvůru

zabije, vezme si její podklady (vytáhne karty z balíčku poklady do své ruky) a pokračuje další fází kola. Není-li hráč schopný přemoci nestvůru svépomocí, může požádat o pomoc ostatní hráče, většinou přislíbením získání pokladů nebo jiné odměny.

Vyhraje-li souboj nestvůra, je hráč nucen k útěku. Útěk je považován za úspěšný, hodí-li hráč na hrací kostce pětku či šestku, boj je u konce a pokračuje se další fází kola. Pokud se ovšem nepodaří hráči z boje utéct, děje se mu „zlořádstvo“ jenž je specifikováno samotnou kartou nestvůry [2].

Kompletní pravidla lze nalézt v balení hry nebo v dokumentu uvedeném na oficiálních webových stránkách `http://www.worldofmunchkin.com/rules/munchkin_rules.pdf`.

3 Základní koncepce

3.1 Architektura komunikace

Možnosti architektury, na kterých jsou realizovány všechny síťové aplikace, lze rozdělit do dvou základních skupin. Jednou z nich je klient – klient, často označované zkratkou P2P odvozenou z anglického Peer-to-Peer. Charakteristickou vlastností této architektury je rovnost postavení a odpovědnosti koncových bodů [3]. P2P model využívá decentralizované správy, kde každý koncový bod může iniciovat požadavky na ostatní koncové body simultánně s odpověďmi na požadavky přijaté [4].

Naopak architektura, kde jeden nebo více koncových bodů vysílá požadavky na centralizovaný bod a přijímá jeho odpovědi, se nazývá klient – server [5]. Klient nevykonává funkcionalitu akce samotné, pouze vyšle požadavek na server, ten jej zpracuje, vykoná a odešle výsledek zpět na klienta pro jeho zobrazení či jiné zpracování.

Hybridní model architektury stojí mezi těmito dvěma základními. Ukrývá v sobě centralizovaný bod, který pomáhá klientům najít ostatní klienty a dorozumět se mezi sebou. Nabízí rozhraní pro vyměňování poskytované funkcionality vycházející z architektury klient-server a zachovává rovnost klientů typickou pro Peer-to-Peer sítě [14].

3.2 Softwarová architektura

Moderní aplikace často bývají rozloženy do vrstev, které jsou na sobě nezávislé, aby bylo možné tyto vrstvy vyměňovat za jiné, nebo je modifikovat bez obavy narušení ostatních částí programu. V mém případě musí tuto vlastnost implementace splňovat už jenom z toho důvodu, že budu potřebovat realizovat hráče s jednoduchou umělou inteligencí, kterému je například uživatelské rozhraní zbytečné. Nabízí se návrh pomocí třívrstvé architektury.

Třívrstvá architektura je jedním z druhů vícevrstvé architektury pro klient-sever aplikace. Prezentační vrstva aplikace reprezentující uživatelské rozhraní je oddělena od doménové logiky představovanou druhou vrstvou, tzv. logickou, a od vrstvy třetí, umožňující přístup k datům, nazývanou datovou [15].

Podrobnější informace k reprezentaci prezentační vrstvy v mém projektu, respektive k realizaci uživatelského rozhraní, lze nalézt v kapitole 6.

Logická vrstva se stará o zpracování příkazů, výpočtů a uskutečňuje rozhodnutí, které charakterizují samotný program. Zároveň manipuluje s daty mezi prezenční a datovou vrstvou. V mém projektu je reprezentována herní logikou, detailnější popis je k nalezení v kapitole 5. Implementace logické vrstvy je zprostředkována doménovým modelem.

Datová vrstva je pouze na straně serveru a je realizována třídou načítající data z lokálního úložiště do operační paměti. Klientům jsou data odeslána skrze komunikační spojení.

3.3 Analýza

Akce potřebné k řešení ze strany klienta:

- Nalezení hry
- Připojení ke hře
- Komunikace s protihráči
- Zjištění prekondicí k uskutečnění akce v rámci herní logiky
- Vykonání akce
- Zpracování vzniklých následků

Pro projekt hry Munchkin v prostředí internetu jsem využil modelu architektury klient-server, protože server řídí běh celé hry, mění tah hráčů, fáze kol, zprostředkovává centralizovaný bod pro všechny druhy komunikace apod. Klienti jsou robustní desktopové aplikace (tlustý klient), které šetří režijní náklady na komunikaci tím, že implementují většinu herní logiky.

Zároveň bude implementována třívrstvá architektura softwaru, přičemž datová vrstva bude reprezentována lokálním úložištěm a bude sloužit pouze k prvotní inicializaci.

3.4 Implementace

3.4.1 Technologie

Jak na straně klienta, tak na straně serveru stojí aplikace postavená na platformě Java využívající technologie pro přenos dat pomocí internetu – konkrétně sockety. Zprostředkované grafické uživatelské rozhraní je realizováno technologií swing. K tvorbě všech prvků uživatelského rozhraní nebyl použit žádný nástroj pro automatické generování výsledného zdrojového kódu. Swing jsem podrobněji popsal v kapitole 6. Pro tvorbu veškerých diagramů obsažených v této práci jsem použil online nástroj pro návrh diagramů na stránkách `creately.com`.

3.4.1.1 WebStart Klientská část aplikace navíc měla využívat technologii WebStart pro své spuštění. Java Web Start poskytuje plnohodnotným Java aplikacím možnost být spuštěny jedním kliknutím skrze webové rozhraní. Uživatelé si mohou spustit aplikace bez obtěžujícího instalačního procesu [7]. Zmíněné kliknutí se provede na odkaz umístěný na webových stránkách. Jde o odkaz na soubor typu JNLP (Java Network Launch Protocol), který zajistí stažení aplikace, její načtení do vyrovnávací paměti a následné spuštění.

3.4.1.1.1 Výhody

- Aplikace je spuštěna bez potřeby instalace
- Aplikace je udržována v lokální mezipaměti pro zvýšení výkonu

- Aktualizace aplikací se stahují automaticky při spuštění
- Platformě nezávislá aplikace (vychází ze samotné podstaty platformy Java)

3.4.1.1.2 Integrace hry Munchkin do WebStart Protože Java WebStart podporuje přímé spuštění aplikací bez instalačního procesu, jsou pro tyto aplikace povoleny pouze minimální práva. Aby tato technologie zajistila ochranu před potenciálně nebezpečnými softwary, vyžaduje povolení ve formě podepsaných JAR souborů. Při stahování takového souboru se zobrazí uživateli informace o certifikátu, použitém k jeho podepsání a dotáže se uživatele, zda mu chce důvěřovat či nikoliv.

O správu certifikátu by se měla starat serverová část, ovšem nastal problém se získáním certifikátu. Po dohodě s vedoucím práce byla integrace hry Munchkin do technologie WebStart vypuštěna z náplně bakalářské práce.

3.4.2 Využité návrhové vzory

3.4.2.1 Decorator (Dekorátor) Návrhový vzor dekorátor je všeobecně znám také pod názvem Wrapper (z anglického „to wrap“ – obalovat), což napovídá o jeho významu. Dynamicky připojuje objektům další zodpovědnosti. *Dekorátor poskytuje flexibilní alternativu k dědičnosti používané za účelem rozšíření funkcionality* [13]. Při využívání principu dědičnosti, jsou přidávány funkcionality každému objektu vytvořeného pomocí potomka. Naopak pomocí dekorátoru můžeme přidat funkcionalitu jenom objektům, u kterých je to vyžadováno, a to tím způsobem, že obalíme objekt jiným objektem, jenž funkcionalitu přidává.

Výše popisovaný návrhový vzor jsem osobně neimplementoval, ale využil jsem jej například u objektů knihoven .net a .io, kde se o implementaci postarali již tvůrci těchto balíků. Jeden z příkladů je možno nalézt v kapitole 7.5.2. Na chvíli se pozastavím u implementace dekorátoru u objektů knihovny .io, konkrétně budu popisovat příklad využití u OutputStreamu. Třída OutputStream je abstraktní třída, kde všichni její potomci reprezentují výstupní stream bytů [7]. Všechny konkrétní implementace OutputStreamu, jako například BufferedOutputStream, ObjectOutputStream a FileOutputStream, mají konstruktor, který přijímá jako parametr instanci konkrétní implementace stejné abstraktní třídy, což může být považováno za poznávací znak dekorátoru.

Chci-li zapisovat data do souboru, můžu použít FileOutputStream.

```
file = new File(”c:/soubor.txt”);
FileOutputStream fos = new FileOutputStream(file);
```

Chci-li přidat funkcionalitu BufferedOutputStreamu, stačí objekt reprezentující FileOutputStream objektem této třídy obalit („dekorovat“).

```
BufferedOutputStream bos = new BufferedOutputStream(fos);
```

Analogicky lze pokračovat pro všechny ostatní konkrétní implementace třídy OutputStream.

3.4.2.2 Observer(Pozorovatel) *Definuje závislosti mezi objekty v poměru 1:N tak, že kdykoliv změní objekt (1) svůj stav, všechny jeho závislosti (N) budou automaticky aktualizovány* [13]. V návrhovém vzoru Observer je objektem, který spravuje seznam jeho závislostí nazýván subject (česky označován stejně jako celý vzor - pozorovatel). Samotné závislosti jsou nazývány observer (posluchač). Automatické aktualizování posluchačů je realizováno jednou z jejich metod volanou pozorovatelem.

V programovacím jazyce Java lze k implementaci využít rozhraní. Následující příklad popisuje situaci, kterou jsem v rámci realizace skutečně řešil. Sleduje se stav hrdiny. Při každé změně vyšle pozorovatel (hrdina - Hero) zprávu skrze metodu implementovanou rohraním (například levelChanged) všem svým posluchačům. Ti ji zachytí a reagují na ni. Například samotná hra nepřímo reaguje aktualizací uživatelského rozhraní a oznámením skutečnosti ostatním hráčům.

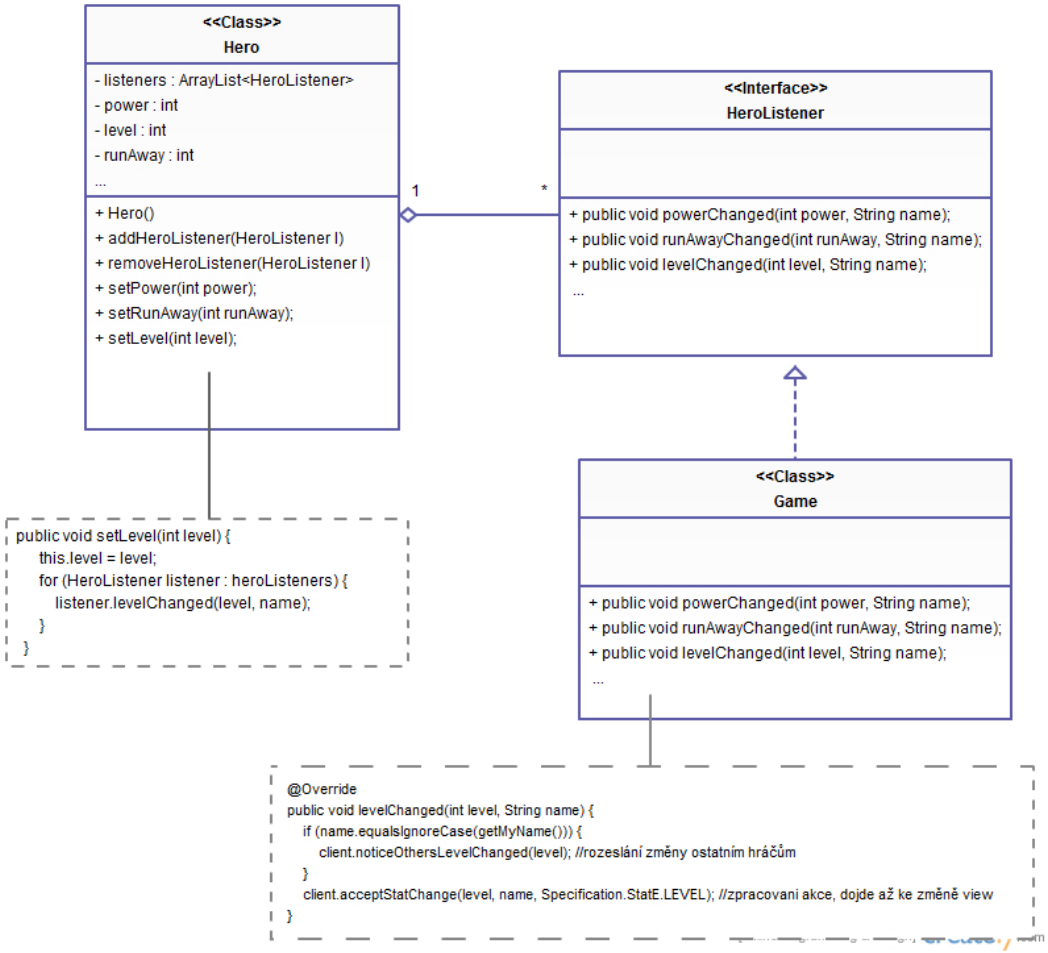
Ve svém projektu jsem tento návrhový vzor použil dále například při pozorování změny stavu hry, kde konkrétním příkladem může být připojení nového hráče do hry, nebo pro sledování změny fáze hry a na mnoha jiných místech. Další konkrétní příklady lze nalézt v kapitole 5.

3.4.2.3 Singleton (Jedináček) *Zajišťuje, že třída má pouze jednu instanci, a poskytuje k ní globální přístup* [13]. Abychom dodrželi definici jedináčka, je zapotřebí udělat jeho reprezentativní třídu zodpovědnou za udržování jeho jediné instance a zajistit, aby poskytovala přístup k této instanci.

Daných požadavků lze dosáhnout vytvořením třídy s privátní statickou proměnou typu třídy samotné. Konstruktor této třídy bude také privátní a bude volán jen v případě, že instance ještě není vytvořena a udržována ve výše popisované proměnné.

Ve svém projektu jsem využil jedináčka k reprezentaci hrací kostky. Hrací kostka generuje a vrací náhodná čísla od 1 do 6 metodou rollADice(). Je schopna také vrátit poslední hodnotu hozenou na kostce pomocí getLastRollValue() a hodnotu kostky na opačné straně, než je hozená hodnota, metodou getFlippedDiceRoll().

```
public class Dice {

    private final Random random;
    private int rollValue;
    private static Dice dice;

    private Dice(){
        random = new Random(System.currentTimeMillis());
    }

    public static Dice getDice(){
        if (dice == null){
            dice = new Dice();
        }
```



Obrázek 1: Návrhový vzor pozorovatel

```
        return dice;
    }

    public int rollADice() {
        rollValue = random.nextInt(6) + 1;

        return rollValue;
    }

    public int getLastRollValue(){ return rollValue; }

    public int getFlippedDiceRollValue(){ return 7-rollValue; }
}
```

Výpis 1: Implementace hrací kostky

4 Karty

4.1 Analýza

4.1.1 Karty

Jak bylo v pravidlech hry zmíněno, karty se dělí na dvě základní skupiny - poklady (Treasure) a dveře (Door). Mezi poklady patří předměty (Item), karty zvyšující úroveň hráče (Go Up a Level), karty umožňující zesílit buď samotné hráče, nebo monstra stojící proti nim (Bonus), a speciální poklady, jejichž úkolem je upravit právě probíhající akci (boj, útěk, hod kostkou, apod.) ve svůj prospěch (Special Treasure) [2]. Předměty se dále dělí dle možnosti využití na helmy (Headgear), brnění (Armor), boty (Footgear), zbraně (Weapon).

Nestvůry (Monster), kletby (Curse), rasy (Race), povolání (Class), karty upravující sílu jednotlivých monster (Monster Enhancer) a speciální dveře upravující podobu hráče, nebo ovlivňující právě probíhající akci v hráčův neprospěch (Special Door), patří mezi dveře [2].

4.1.2 Balíčky

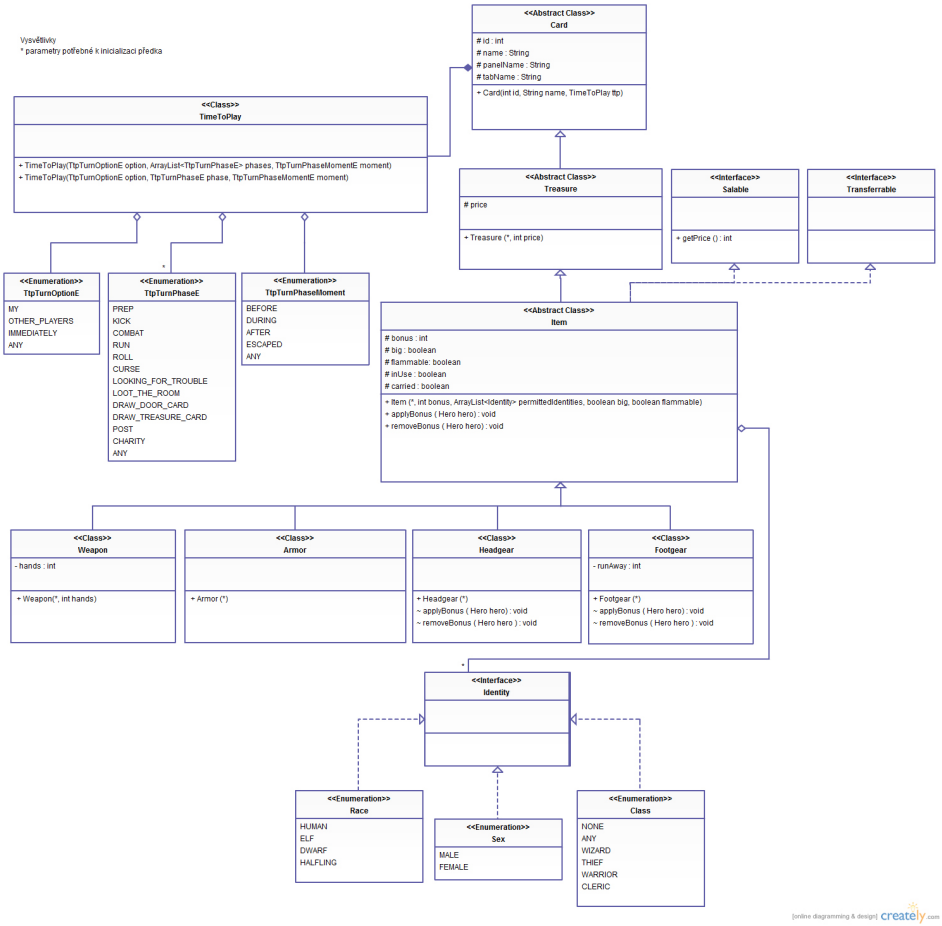
Než se karty dostanou k hráči, musí být rozděleny do balíčku, ze kterých je možné je získat. Vyřazené karty ze hry je také potřeba udržovat pohromadě v samostatném balíčku. V balíčcích bude možno karty zamíchat, nebo je z nich vybrat, či naopak je tam vložit.

4.2 Implementace

4.2.1 Karty

Na základě logického rozdělení karet dle analýzy jsem vytvořil třídy reprezentující právě tyto entity. Pro řešení tohoto problému jsem s výhodou využil dědičnosti. Společné vlastnosti karet jsou implementovány do předků a specifické vlastnosti do potomků. Určení vlastností entit bylo předmětem podrobné analýzy, ze které jako výsledek vzešel soubor typu xls, jenž je součástí přílohy CD. Ze souboru byla vybrána data pro jednotlivé entity a vložena do zvláště uložených souborů pojmenovaných po samotných entitách. Zmíněné soubory jsou textového typu (txt) a rovněž jsou součástí přílohy CD.

Realizace implementace je znázorněna třídnímy diagramy. Následuje ukázka implementace karet předmětů (Item). Kompletní přehled třídních diagramů je k nalezení v tištěných přílohách nebo na přiloženém CD pro lepší čitelnost.



Obrázek 2: Třídní diagram karet předmětů

4.2.2 Parsování

K převodu karet z předpřipravených textových souborů jsem využil parsování pomocí StringTokenizeru. StringTokenizer je třída umožňující zpracovávat textové řetězce jeho rozložením na části zvané tokeny. Znaky oddělující jednotlivé tokeny lze specifikovat v konstruktoru objektu přestavující StringTokenizer, tedy v době vytvoření, nebo po vytvoření objektu při samotném zpracovávání tokenu. Protože moje interpretace karet v textových souborech odděluje jednotlivé vlastnosti karty vždy stejným znakem – tabulátorem – využil jsem první možnost, tedy specifikaci znaku oddělující tokeny při vytváření objektu. Další možností vytvoření objektu třídy StringTokenizer, vedle zadání zpracovávaného textu a oddělovače, je včetně upřesnění, zda se bude oddělovač samotný považovat za token, či nikoliv [7].

StringTokenizer implementuje rozhraní Enumeration⟨E⟩, které umožňuje generování posloupnosti elementů a jejich vrácení metodou nextElement [7] interpretovaného StringTokenizerem také jako nextToken vracející textový řětězec.

Každým tokenem byl inicializován objekt, který se využil pro vytvoření výsledné karty, repsektive instance třídy Card.

```
public class Deck {

    private Stack<Treasure> treasures;
    private URL resource;
    private BufferedReader in;
    private StringTokenizer st;
    private String line;

    private void parseArmor() throws FileNotFoundException {

        resource = getClass().getResource("armor.txt");
        in = new BufferedReader(new FileReader(new File(resource.getPath())));

        String name;
        Integer id, price, bonus;
        Boolean big, flammable;
        ArrayList<Identity> permittedIdentities;
        TimeToPlay ttp;

        try {
            while ((line = in.readLine()) != null) {
                st = new StringTokenizer(line, "\t");
                id = Integer.parseInt(st.nextToken());
                name = st.nextToken();
                price = Integer.parseInt(st.nextToken());
                bonus = Integer.parseInt(st.nextToken());
                big = Boolean.parseBoolean(st.nextToken());
                String[] identities = st.nextToken().split(",");
                permittedIdentities = parseIdentities(identities);
                flammable = Boolean.parseBoolean(st.nextToken());
                ttp = parseTimeToPlay();

                treasures.push(new Armor(id, name, price, bonus, big, permittedIdentities, flammable,
                        ttp));
            }
        } catch (IOException ex) {
            System.out.println("Error while parsing ARMOR cards");
            System.out.println(ex);
        }
    }
}
```

Výpis 2: Práce s třídou StringTokenizer

4.2.3 Balíčky

Pro reprezentaci balíčků karet se nejlépe hodí objekt zásobníku, protože mechanismus LIFO (last-in-first-out) přesně kopíruje reálnou manipulaci s balíčky. Karty se vybírají vždy z vrchu balíčku, kde je umístěna poslední přidaná karta.

5 Herní logika

S kartami a pravidly hry úzce souvisí herní logika. Jednou z otázek řešenou herní logikou je, je-li akce, kterou vyžaduje hráč provést, v rámci pravidel hry. Častým případem je ověřování, zda lze zahrát kartu vybranou hráčem. Kontrola se provádí pomocí porovnání aktuální fáze hry a vlastností karty určující, kdy je kartu možno zahrát.

Herní logika dále řeší dopad provedené akce na hru samotnou. Budu-li pokračovat ve výše popisovaném příkladě, je nutné řešit efekt karty na hráče, jenž kartu zahrál, efekt na ostatní protihráče a dopad na vývoj hry.

Akce

- Vytažení karty z balíčku
- Zahrání karty
- Odhození karty
- Vrh kostkou
- Pomoc
- Prodej karet (předmětů)

Dopady

- Změna fáze hry
- Změna stavu hrdiny (úroveň, síla, bonus k útěku, rasa, povolání, pohlaví)
- Změna stavu nestvůry (síla, poklady)
- Změna počtu a rozložení karet

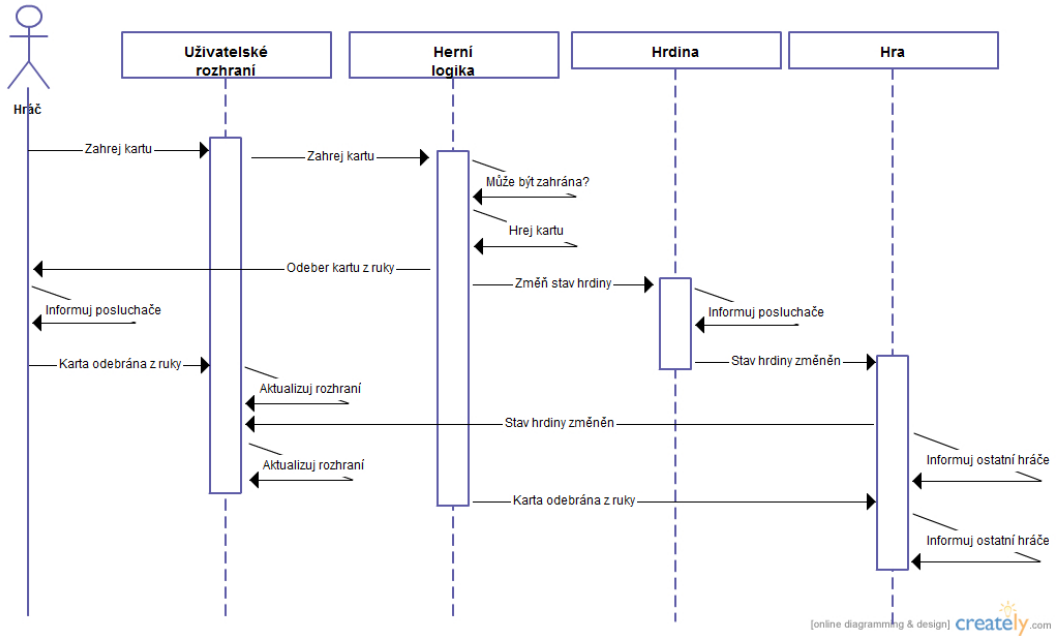
Pro sledování akcí a vytvoření důsledků jsem v hojné míře využíval návrhový vzor pozorovatel 3.4.2.2. Prováděné akce jsou události vykonané na pozorovateli a dopady jsou konkrétní reakce na vyvolané události uvnitř posluchačů. V několika následujících odstavcích se budu držet příkladu provedené akce „Zahrání karty“ a jejích dopadů „Změna stavu hrdiny“ a „Změna počtu a rozložení karet“.

Hráč provede akci „Zahrání karty“ kliknutím levého tlačítka myši na část GUI, konkrétně na ImagePanel reprezentující kartu (více o implementaci GUI a ImagePanelu v kapitole 6). Událost je zpracována posluchačem, kde se herní logice předává karta, na které byla událost vyvolána. Herní logika se ujistí, zda karta může být zahrána, protože je to v rámci její kompetence. Budu pokračovat ve scénáři, ve kterém je kartu možno hrát.

Dopady této akce jsou také v rámci kompetence herní logiky. Aplikuje dopad karty změnou stavu hrdiny (například zvýšením síly). Hrdina je implementován jako pozorovatel, tudíž při změně síly oznámí všem svým posluchačům, že k této skutečnosti došlo. Posluchačem hrdiny (implementačně HeroListener) je samotná hra. Ta v rámci zpracování

události se postará o rozeslání provedené akce ostatním hráčům a aktualizaci uživatelského rozhraní.

Se zahráním karty je většinou spojen také dopad na rozložení karet ve hře či na jejich počet. Po odehrání karty se karta může přemístit v rámci hrací plochy nebo mezi hráčem a hrací plochou. Jedná-li se o pohyb karet v rámci stolu, řeší oznámení o změně pozice karty hra. Je-li s kartou manipulováno mezi hráčem a hrací plochou, je událost zpracována pozorovatelem hráče, protože právě on v sobě udržuje informace o kartách v ruce či na stole (ale už neřeší kde na stole). Posluchačem hráče je hra, která na tuto událost zareaguje upozorněním ostatních hráčů. Každý z hráčů následně přemístí protihráčovu kartu ve svém uživatelském rozhraní na patřičné místo a aktualizuje počet protihráčových karet v ruce či ve hře (pokud je to třeba). Popisovanou akci i s jejími dopady zachycuje následující sekvenční diagram. Karta se přemisťuje mezi hráčm a hrací plochou.



Obrázek 3: Sekvenční diagram akce Zahrání karty

6 Uživatelské rozhraní

Budu-li se zmiňovat o uživatelském rozhraní v následujících odstavcích, budu mít na mysli vždy grafické uživatelské rozhraní (GUI – Graphical User Interface). Uživatelské rozhraní je součást aplikace zprostředkovávající interakce uživatele a programu. Je založeno na základních grafických elementech zvaných komponenty. Ty zobrazují informace a přijímají akce vyvolané uživatelem na tyto komponenty. Následuje charakteristika vybraných nástrojů pro práci s uživatelským rozhraním na platformě Java.

6.1 AWT

První možnosti práce s GUI na platformě Java nabízí nástroj Abstract Windowing Toolkit (AWT), který je založen na komponentách systému, kde běží. Obstarává elementární operace jako zobrazení pohybu myši po obrazovce, zobrazení textu psaného na klávesnici, vykreslování objektů apod. AWT poskytuje realizaci návrhu objektově orientovaného rozhraní pro tyto základní úkony [10]. AWT ovšem nenabízí pouze zpracování těchto nízko-úrovňových požadavků. Dokáže shlukovat komponenty do kontejnerů a tak spravuje návrhy uživatelského rozhraní. AWT zprostředkovává devět základních komponent jako například Button, Label, Checkbox, Canvas a kontejnery jako Window, Frame, Dialog a Panel [10]

6.2 SWT

Standard Widget Toolkit (SWT) je nástroj pro návrhy uživatelského rozhraní, který byl původně vyvinut firmou IBM. Využívá nativní knihovny operačního systému za použití JNI (Java Native Interface). Dokáže napodobit vzhled komponent a kontejnerů operačního systému, na kterém běží, protože využívá systémově závislá aplikační rozhraní [16].

6.3 Swing

Swing je knihovna, jež je součástí JFC (Java Foundation Classes) společnosti Oracle. Vychází z knihovny AWT a je postavena na MVC (Model-View-Controller) architektuře. Protože je to knihovna, kterou jsem ve svém projektu využil nejvíce, popíšu podrobněji některé její vlastnosti.

6.3.1 Komponenty

Jak již bylo uvedeno, jednotkou uživatelského rozhraní je komponenta. Komponenta je abstraktní objekt reprezentující grafické zpracování. Stěžejní třídou pro všechny swing komponenty je JComponent [7]. Mezi nejčastěji využívané komponenty patří:

JButton tlačítko

JLabel textový popisek

JTextField jednořádkové textové pole

JTextArea víceřádkové textové pole

JComboBox komponenta pro výběr možnosti z drop-down listu

JChackBox zaškrtávací políčko

6.3.2 Kontejnery

Kontejnery jsou speciálními typy komponent, které jsou schopny udržovat další komponenty. Komponenty přidávané do kontejnerů jsou udržovány v seznamu, přičemž lze definovat pozici v tomto seznamu při vkládání komponenty indexem. Není-li index definován, je komponenta přidána na konec seznamu [8]. Zajímavostí může být, že JComponent je potomkem třídy Container (třída knihovny AWT), jenž je potomkem třídy reprezentující AWT komponenty – Component. Pro lepší zorientování může pomoci fakt, že kontejnery nemají definován žádný vnější vzhled kromě barvy pozadí a vyhraněné oblasti na obrazovce.

Existují dva typy swing kontejnerů. JPanel a takzvané content pane (volně přeloženo jako obsahová tabule). Instance třídy JPanel jsou vytvářeny klasicky bezparametrickým či parametrickým konstruktorem. Na rozdíl content pane lze získat get metodou z oken jako JFrame, JWindow, JDialog a dalších [8].

JPanel jsem při vývoji uživatelského rozhraní využíval téměř v každé části view. Nalézt ho tedy můžete například v každém view s názvem končícím JPanel nebo v hlavním okně sdružující všechny jednotlivé view – ClientView.

Content pane jsem používal zejména k nastavování pozadí u oken typu JFrame a JDialog. Práci s ním lze dohledat například v ClientView nebo v ukázce implementace části uživatelského rozhraní na konci této kapitoly.

6.3.3 Správa rozložení

Nedílnou součástí grafického designu je správné rozvržení objektů ve výsledném uživatelském rozhraní. Pro knihovnu swing (ale také pro AWT [7]) představuje rozhraní LayoutManager možnost, jak s rozložením manipulovat. Přestože komponentám může být přidělena velikost či zarovnání individuálně, je to právě LayoutManager, který má poslední slovo, když jde o tyto hodnoty v rámci kontejneru [7]. Při zhotovování uživatelského rozhraní jsem se setkal s řadou různých implementací LayoutManageru, následuje popis některých z nich.

6.3.3.1 BorderLayout Pokládá komponenty, přičemž manipuluje s jejich rozměry a přemisťuje je tak, aby se vlezly do pěti oblastí definovaných BorderLayoutem. Žádná z oblastí nemůže obsahovat více než jeden komponentu. Oblasti jsou definovány jako veřejné statické konstanty třídy BorderLayout. Jejich hodnoty jsou ukryty pod názvy NORTH, SOUTH, EAST, WEST a CENTER, které napovídají, kam jsou komponenty pokládány [7]. BorderLayout jsem použil například v části uživatelského rozhraní pokrývající view pro textový chat (ChatJPanel).

6.3.3.2 GridLayout Správce rozložení, který umisťuje komponenty do mřížky obdélníkového tvaru, se nazývá GridLayout. Všechny buňky dané mřížky mají stejný rozměr. Při konstrukci objektu GridLayout je možné definovat počet řádků a sloupců mřížky, přičemž komponenty se vkládají do mřížky zleva doprava, shora dolů. [7]. Výše popisovaného správce rozložení jsem použil například ve view udržujícím stavy hrdinů a monster (FightCalcJPanel).

6.3.3.3 FlowLayout Umisťuje komponenty vedle sebe ve směru, který je definován jednou z konstant LEFT_TO_RIGHT či RIGHT_TO_LEFT. Lze také upřesnit zarovnání komponent vlastností align (nabývá hodnot LEFT, RIGHT, CENTER, LEADING, TRAILING). Většinou se FlowLayout využívá k uspořádání tlačítek v panelu [7]. Já jsem jej využil například k zarovnání komponent starajících se o zobrazení kartičky v rámci panelu (CardsInGameJPanel, CardsInHandJPanel, DecksJPanel).

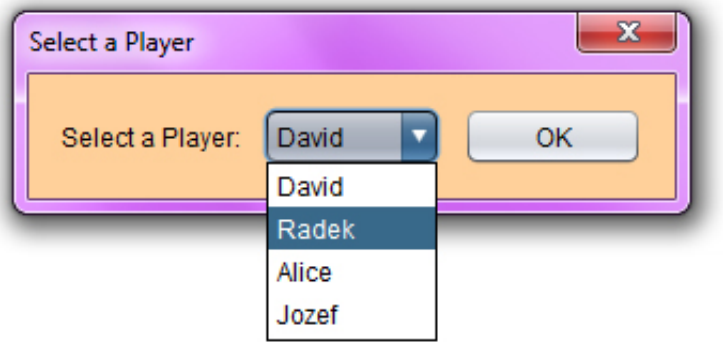
6.3.4 Chování

Princip ošetřování chování je založen na návrhovém vzoru Pozorovatel, jenž je blíže popsán v kapitole 3.4.2.2.

Následuje ukázka práce se swing komponentami, konkrétně jde o ukázku použití návrhů rozložení, včetně implementace ošetření chování. Jedná se o mnou implementovanou část uživatelského rozhraní využívanou ve hře, sloužící pro výběr hodnoty ze seznamu. Do konstruktoru této části GUI se kromě okna, ze kterého je dialog vyvolán, zadává, zda má být dialog zviditelněn. Vlastními parametry je textová proměnná upřesňující co se bude ze seznamu vybírat a kolekce možností.

```
public class SelectDialog extends javax.swing.JDialog {

    private JButton confirmButton;
    private JComboBox valuesCB;
    private String  selectedValue;

    public SelectDialog(java.awt.Frame parent, boolean modal, String object, Vector values) {
        super(parent, "Select a~" + object  ,modal);
        initComponents();

        this.setLayout(new GridBagLayout());
        this.getContentPane().setBackground(new Color(255, 208, 154));

        JPanel panel = new JPanel();
        panel.setBackground((new Color(255, 208, 154)));
        panel.setBorder(new EmptyBorder(10, 10, 10, 10));
        panel.setLayout(new GridLayout(1, 2, 10, 10));

        panel.add(new JLabel("Select a~" + object + ":"));

        final DefaultComboBoxModel model = new DefaultComboBoxModel(values);
        valuesCB = new JComboBox(model);

        panel.add(valuesCB);
        confirmButton = new JButton("OK");
        panel.add(confirmButton);

        confirmButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                    selectedValue = (String)  valuesCB.getSelectedItem();
                    dispose();
            }
        });

        this.add(panel);
        this.pack();
        this.setResizable(false);
        this.setLocation((java.awt.Toolkit.getDefaultToolkit().getScreenSize().width/2)-(this.
            getWidth()/2),
                    (java.awt.Toolkit.getDefaultToolkit().getScreenSize().height/2)-(this.getHeight
                        ()/2));
        this.setVisible(modal);
    }

    public String  getSelectedValue(){
            return this.selectedValue;
    }
}
```

Výpis 3: Dialog pro výběr hodnoty z drop-down listu specifikovaného za běhu programu



Obrázek 4: Dialog pro výběr protihráče

6.3.5 Serverová část

Pro jednoduché testování mého projektu je uživatelské rozhraní také na straně serveru, ve kterém se zadají základní parametry hry. Toto GUI bude nahrazeno funkcionalitou portálu, kde si klient nastaví jakou hru chce hrát, pošle výběr na portál, který už samotnou hru spustí i s příslušnými parametry.

7 Komunikace

Koncové body komunikačních uzlů si mezi sebou vyměňují zprávy, v případě tohoto projektu může jít například o kartičky či jiné typy charakterizované v kapitole 7.4. Fyzická komunikace mezi dvěma zařízeními je ovšem velmi nízko-úrovňová činnost. Jak převést objekt představující kartičku, se všemi jejími parametry na jiné zařízení nebo lokální úložiště a případně jej v budoucnu znovu používat jako objekt? K tomu lze s výhodou využít proces zvaný serializace a deserializace.

7.1 Serializace

Serializace je schopnost konvertovat objekty uložené v paměti do podoby, ve které je možné je posílat byte po bytu do cíle. Často je potřeba přenesené objekty zpracovat či odeslat v jiném prostředí na jiném zařízení, než ze kterého byly odeslány, proto je tento proces nezávislý na zpracovávajícím prostředí [12]. V rámci procesu serializace objektu v prostředí Javy, je zakódována jeho třída včetně jejího jména, signatury, všech referencí a nestatických proměnných neoznačených signaturou transient [7].

7.2 Deserializace

Proces rekonstrukce přenesených serializovaných dat, při kterém dochází k znovuvytvoření sémanticky identického klonu originálního serializovaného objektu se nazývá deserializace [9].

7.3 Rozhraní Serializable a Externalizable

Jedním z řešení jak umožnit serializaci objektů v platformě Java je implementovat rozhraní Serializable danou třídou. Všichni potomci třídy implementující toto rozhraní budou také serializovatelné [7]. Zvláštností může být, že popisované rozhraní nemá žádná pole ani metody, které by museli být třídami implementovány, jak je zvyklé u běžných rozhraní. Jeho funkce spočívá pouze v označení možnosti býti serializován.

Aby bylo možné serializovat potomky neserializovatelných tříd, potomek musí převzít zodpovědnost za ukládání a obnovování public, protected a package proměnných. Potomek přebírá tuto zodpovědnost pouze v případě, je-li předkem implementován bezparametrický konstruktor, který umožní inicializaci jeho stavu. Zmíněný konstruktor je využit během procesu deserializace [7].

Není-li k dispozici třída reprezentující deserializovaný objekt, je vyvolána výjimka ClassNotFoundException. Kromě existence třídy je ale také zapotřebí zajistit, že třída je kompatibilní se třídou, která byla použita pro vytvoření instance přenášeného objektu. Právě k tomuto účelu slouží číslo verze zvané serialVersionUID [7].

Vedle implementace rozhraní Serializable je možné implementovat i jiné rozhraní, které zajistí stejnou funkcionalitu. Jedná se o rozhraní Externalizable. Ve skutečnosti rozšiřuje rozhraní Serializable, zde už je ovšem potřeba implementovat metody readExternal a writeExternal konkretizující jak data serializovat [7].

7.4 Analýza

Z podstaty projektu vyplývá, že komunikace mezi serverem a klientem bude probíhat v prostředí internetu. Mezi serverem a klientem se budou vyměňovat zprávy následujících typů:

- Textové zprávy
  - chatové
  - informace o průběhu hry
- Číselné zprávy
  - charakteristiky hráčů a nestvůr (úroveň, síla, apod.)
  - hodnota vrhu kostkou
- Vlastní objektové zprávy
  - karty
  - fáze kola
- Hlas

7.4.1 Vyhodnocení

Serverová i klientská aplikace bude potřebovat pro vzájemnou komunikaci dvě komunikační spojení. Jedno spojení bude představovat obousměrný objektový stream pro přenos objektů specifikovaných koncovými stranami, realizovaný na protokolu TCP. Bližší charakteristika přenášených dat bude součástí přeneseného objektu. Z toho vyplývá, že je potřeba implementovat vlastní protokol pro komunikaci a interpretaci přenášených objektů.

Další komunikační spojení je potřebné pro přenos hlasových zpráv. Protože není potřeba spojitého, spolehlivého přenosu, bude realizován nad protokolem UDP. Komunikační spojení bude realizováno obousměrným streamem umožňujícím přenos objektů vzniklých technologiemi pro zpracování hlasu skrze datagramy.

7.5 Implementace

Pro implementaci v technologii Java se pro vzájemnou komunikaci dle výše kladených požadavků nejlépe hodí využití socketů. *Socket je jedním z koncových bodů obousměrné komunikační linky mezi programy běžícími v síti* [7]. Socket má přiřazen číslo portu, takže TPC vrstva může identifikovat aplikaci, které byla data zaslána.

7.5.1 Komunikace nad TCP

V platformě Java je to balík java.net, který poskytuje třídu Socket realizující socket na straně klienta a ServerSocket představující socket na straně serveru, který je schopný naslouchat a přijímat požadavky o spojení klientů [7].

7.5.1.1 Klient Pro vytvoření a připojení socketu na straně klienta je potřeba znát adresu, na které běží server a port, na kterém čeká na příchozí spojení. Tyto údaje získám z části uživatelského rozhraní starající se o připojení hráče do hry.

Pokud existuje hostitel na dané adrese a naslouchá na specifikovaném portu, naváže se spojení. Vytvořený socket se předá třídě reprezentující hráče (Player) jako parametr konstruktoru.

Player si socket uloží do třídní proměnné a využije jej k vytvoření streamů pro komunikaci, jeden pro směr odchozí a jeden pro směr příchozí.

Využití odchozího streamu řídí klient sám odesláním zpráv či nikoliv, ale příchozí stream je potřeba sledovat nepřetržitě, protože neví, kdy byla zpráva ze serveru poslána. Proto běží sledování příchozího streamu ve smyčce po celou dobu trvání spojení.

Aby bylo možné pokračovat v běhu programu, je nutné provádět toto čtení v samostatném vlákně, mimo hlavní běh programu. Implementaci nového vlákna jsem realizoval rozhraním Runnable využívaného třídou Player.

7.5.1.2 Server K vytvoření ServerSocketu je potřeba určit port, na kterém bude server naslouchat a přijímat spojení klientů. Tento údaj získám z části uživatelského rozhraní na straně severu (správně by se o to měl starat herní portál, více o důvodech implementace GUI pro serverovou část lze najít v kapitole 8). Poté, co je socket vytvořen, může se začít s čekáním na příchozí žádosti o navázání spojení. U každého z připojených hráčů se uloží jeho socket, získaný metodou

public Socket accept() throws IOException

třídy ServerSocekt. Následně se inicializují streamy a sleduje se stav příchozího streamu, stejně jako na straně klienta v samostatném vlákně po dobu trvání spojení. Naslouchání na žádosti o navázání spojení je ukončeno v momentě, kdy je k serveru připojeno tolik hráčů, kolik bylo nastaveno v parametrech hry ve zmíněné části GUI.

7.5.2 Streamy pro přenos objektů

7.5.2.1 Odchozí stream Pro implementaci jsem zvolil objektový stream, který je v platformě Java představován třídou ObjectOutputStream. Zapisuje primitivní datové typy a grafy Java objektů do odchozího streamu. Objekty zapisované do tohoto streamu musejí implementovat rozhraní java.io.Serializable nebo java.io.Externalizable.

Protože do streamu potřebuji zapisovat blíže nespecifikované objekty, využívám metodu

public final void writeObject(Object obj) throws IOException

, kde se jako parametr předává objekt k zapsání do streamu.

V případě zapsání neserializovatelného objektu do streamu se vyvolá vyjímka NotSerializableException, ta je zachycena ObjectOutputStreamem a přeruší proces zápisu do streamu [7].

Parametrem konstruktoru třídy ObjectOutputStream je OutputStream, který získám z klientského socketu. Abych zajistil plynulý běh zápisu do výstupního streamu, obalil jsem OutputStream do instance třídy BufferedOutputStream. BufferedOutputStream zapisuje data do OutputStreamu bez potřeby získávat zdroje k potřebnému OutputStreamu pro každý zapsaný byte [7].

7.5.2.2 Příchozí stream Na druhé straně komunikačního spojení, kde je zdrojovým streamem ObjectOutputStreamu, stojí ObjectInputStream. Vedle toho, že obnovuje objekty serializované ObjectOutputStreamem, kontroluje také, že se všechny typy objektu grafu vytvořeného streamem, shodují s třídami přítomnými v Java Virtual Machine. V případě, že tomu tak není, vyvolá se výjimka ClassNotFoundException. Proto je potřeba mít jak na straně serveru, tak na straně klienta stejné třídy, ve stejné verzi [7].

Ke čtení blíže nespecifikovaného objektu se využívá metoda [7]

public final Object readObject() throws IOException, ClassNotFoundException

Lze číst pouze objekty implementující java.io.Serializable nebo java.io.Externalizable rozhraní.

V mé interpretaci předávám kontruktoru třídy ObjectInputStream jako parametr objekt třídy BufferedInputStream obalující InputStream získaného ze socketu klienta.

7.5.3 Zprávy v objektovém streamu

Objektem přenášeným skrze objektové streamy je instance třídy Message. Message splňuje požadavky na bezpečný přenos implementací rozhraní Serializable.

Proměnné třídy Message

int type typ objektu object

Specification specification specifikace nakládání s objektem object

Object object objekt samotný

String sender jméno odesílatele

ArrayList⟨String⟩receivers jména příjemců

String objectReceiver původní příjemce, bližší charakteristika akce

Každá zpráva, přijatá objektovým streamem, je zpracována metodou

synchronized public void acceptMessage(Player player) instancí třídy Message.

Ta se liší svou implementací podle toho, zda je na serveru či na klientovi, protože server

Tabulka 1: Typy zpráv

Proměnná type	Význam
0	Stav hráče ve hře z pohledu připojení
1	Karta
2	Chat/Připravenost (Klient/Server)
3	Kostka
4	Pomoc
5	Stav hráče ve hře z pohledu kondice (úroveň, síla, apod.)
6	Fáze kola
7	Manipulace s balíčkem odhozených karet
777	Dosažení vítězství
99	Informační zpráva o průběhu hry

nakládá s přijatými zprávami jinak než klient. Server na rozdíl od klienta rozesílá zprávy ostatním účastníkům hry a není potřeba, aby se zprávou nakládal jako hráč.

Aby bylo možné zprávu správně interpretovat, bylo nutné implementovat vlastní protokol.

7.5.3.1 Vlastní komunikační protokol

Pokud dojde ke zpracování zprávy, tak ať už je to klient nebo server, volá se metoda private void interpretMessage(Player player) vyhodnocující další naložení s přijatým objektem. Rozhodující je proměnná type, která je vyhodnocována na přepínači.

7.5.4 Komunikace nad UDP

Oproti komunikaci nad TCP zde neexistuje žádné navázané spojení. Přenos zpráv (datagramů) je nespolehlivý. Odeslané datagramy nejsou druhou stranou potvrzovány, mohou se po přenosové cestě poškodit, změnit pořadí nebo úplně ztratit a to bez povšimnutí příjemce či odesílatele a bez následného požadavku na opakování přenosu. Na druhou stranu by přenos díky ušetření těchto režií měl být rychlejší. Z toho vyplývá, že při komunikaci nad UDP je ze hry i role klienta a serveru, zůstává pouze příjemce a odesílatel, přičemž herní server i herní klient obě tyto role plní, protože herní server přijímá data z voice chatu odeslaná připojenými hráči a rozesílá hráčům ostatním, kteří je přijímají.

Práci s datagramy v platformě Java obstarává třída java.net.DatagramSocket, která umožňuje vytvořit objekt reprezentující socket pro odesílání a přijímání datagramových zpráv [11].

Protože DatagramSocket dokáže jak přijímat tak odesílat datagramové zprávy, dělí se možnosti jak tuto zprávu vytvořit podle využití.

7.5.4.1 Příjemce

Vytvořil jsem nový DatagramSocket za použití parametrického konstruktoru, kde specifikuji, na kterém portu budou data přijímána. Následně byl vytvořen

objekt třídy DatagramPacket, který představuje datagramovou zprávu, konstruktorem pro přijetí.

7.5.4.1.1 Datagramy pro přijetí

Pomocí konstruktoru

public DatagramPacket(byte[] buf, int length)

,kde buf představuje přijímaná data, length délku přijatých dat je možné vytvořit datagram pro přijetí. Bylo možné využít i druhý konstruktor pro vytvoření datagramu pro přijetí, navíc by se musel nastavit parametr int offset, což je odsazení neboli začátek dat našeho zájmu v buf [11].

Přijetí datagramu se realizuje metodou

public synchronized void receive(DatagramPacket p) throws IOException

Kromě toho, že naplní buffer DatagramPacketu daty, získá také informace o odesílateli, konkrétně zdrojovou IP adresu a zdrojový port [11].

7.5.4.2 Odesílatel

Vytvořil jsem nový DatagramSocket za použití bezparametrického konstruktoru, v takovémto případě se otevřel port, který byl přidělen operačním systémem. Následně byl vytvořen objekt třídy DatagramPacket konstruktorem pro odeslání.

7.5.4.2.1 Datagramy pro odeslání

Datagram pro odeslání lze vytvořit pomocí konstruktoru

public DatagramPacket(byte[] buf, int length, InetAddress address, int port)

, kde buf představuje odesílaná data, length délku odesílaných dat, address je cílová adresa protokolu IP a port je cílový port [11]. Ostatní možnosti vytvoření datagramové zprávy byly včetně nastavení offsetu.

Odeslání datagramu se realizuje metodou [11]

public void send(DatagramPacket p) throws IOException

7.5.5 Voice chat

Stejně jako u ostatních druhů komunikací je možné i hlasový chat rozdělit do dvou základních fází. Nejprve je potřeba data získat, zpracovat a odeslat, následně na druhé straně komunikačního kanálu data přijmout, zpracovat a interpretovat. Na této cestě s nimi může být ještě jinak manipulována, například v mém případě do dat zasahoval server, na který data přicházela od všech klietnů.

7.5.5.1 Získání dat

Rozhraní Line reprezentuje jedno nebo více-kanálový zvukový zdroj. Jeden řádek je elementem cesty digitálního zvuku, jako například zvukového mixeru, vstupního nebo výstupního portu, nebo datová cesta do/ze zvukového mixeru [7]. V případě otevření

řádku se rezervují systémové zdroje a naopak při uzavření se zdroje uvolní. Kontrolu lze provádět pomocí metody isOpen() [7].

Rozhraní DataLine rozšiřuje Rozhraní Line. Navíc přidává funkcionalitu spjatou s ovládáním médií jako například oznamování aktuální pozice v médii, ovládání hlasitosti a nastavení datového formátu. Pro interpretaci zvuku je možno použít jedno z rozhraní SourceDataLine nebo Clip, naopak pro vstup rozhraní TargetDataLine [7]. Metoda start() započne sledování I/O streamu.

TargetDataLine je rozhraní rozšiřující DataLine. Poskytuje metodu pro čtení zvuku z vnitřního zásobníku. TargetDataLine může být získán ze zvukového mixeru voláním metody getLine s příslušným DataLine.Info objektem, nebo bez potřeby explicitního získávání mixeru pomocí AutioSystem.getLine, opět s příslušným DataLine.Info objektem [7].

Rozhraní Mixer reprezentuje zařízení s jedním či více řádky (line), protože dokáže smixovat více zdrojů do jednoho výstupu [7].

DataLine.Info objekt specifikuje formát podporovaný DataLine a velikost vnitřního zásobníku pro uchování dat [7]. Formát musí být objektem třídy AudioFormat, ten blíže specifikuje vzorkovací frekvenci, kódování bitů ve zvukových datech, počet kanálů a uspořádání bitů [7].

```
Mixer mixer = null;
if (strMixerName != null) {
    Mixer.Info mixerInfo = getMixerInfo(strMixerName);
    mixer = AudioSystem.getMixer(mixerInfo);
}

private static Mixer.Info getMixerInfo(String strMixerName) {
    Mixer.Info [] mixerInfos = AudioSystem.getMixerInfo();
    for (Mixer.Info mixerInfo : mixerInfos) {
        if (mixerInfo.getName().equals(strMixerName)) {
            return mixerInfo;
        }
    }
    return null;
}
```

Výpis 4: Získání mixeru

7.5.5.1.1 Inicializace TargetDataLine Máme-li specifikovaný objekt mixeru, získáme TargetDataLine z něj, v opačném případě získáme defaultní.

```
DataLine.Info targetInfo = new DataLine.Info(TargetDataLine.class, format, nInternalBufferSize);
if (mixer != null) {
    targetLine = (TargetDataLine) mixer.getLine(targetInfo);
} else {
    targetLine = (TargetDataLine) AudioSystem.getLine(targetInfo);
}
targetLine.open(format, nInternalBufferSize);
```

Výpis 5: Inicializce TargetDataLine

7.5.5.1.2 Spuštění Zároveň se spuštěním sledování I/O streamu, se spustí nové vlákno odesílatele či příjemce.

```
public void start() {
    targetLine.start();
    super.start();
}
```

Výpis 6: Spuštění sledování I/O steamu

Následně stačí data z TargetDataLine číst a odeslat. Data jsou čtena metodou read, která načte data z připraveného I/O streamu a zapíše je do pole bytů, s požadovaným offsetem a délkou.

```
@Override
public void run() {
    try {
        byte[] buffer;
        isRecording = true;
        try (DatagramSocket clientSocket = new DatagramSocket()) {
            while (isRecording && flag) {
                buffer = new byte[322];
                targetLine.read(buffer, 0, 322);
                DatagramPacket sendPacket = new DatagramPacket(buffer, 322, ipAddress, 6001);
                clientSocket.send(sendPacket);
                Thread.sleep(20);
            }
        }
    } catch (IOException | InterruptedException e) {
        System.out.println("Error while recording or sending data");
    }
}
```

Výpis 7: Čtení dat na vstupu a odeslání datagram socketem

7.5.5.2 Odeslání a příjem dat Odesílání a příjem dat je prováděn nad protokolem UDP. Více o této problematice jsem popsal v kapitole Komunikace nad UDP.

7.5.5.3 Zpracování dat Tato úloha je především v roli serveru, který provede součet dat reprezentujících hlas, a odešle je pouze klientům, jejichž hlas není obsažen ve zpracovaném součtu. Abych zajistil plynulost chodu hlasového chatu, jsou na straně serveru otevřeny DatagramSockety pro každého připojeného klienta. Z toho vyplývá, že každý z klientů zasílá svá hlasová data na jiný cílový port, tudíž lze snadno rozlišovat, od kterého klienta data přišla a následně je lze snadněji zpracovat.

7.5.5.4 Interpretace dat Interpretace dat se od jejich získání liší pouze v typu DataLine a metodou realizující jejich zpracování. Na výstupu je použito rozhraní SourceDataLine. Metoda pro zpracování dat na výstup je write, jejíž parametry jsou podobné metodě read u TargetDataLine. Prvním je pole bytů, do něž se budou data zapisovat, další je požadovaný offset a délka zpracovávaných dat.

```
@Override
public void run() {
    try {
        DatagramSocket serverSocket = new DatagramSocket(6002);

        byte[] buffer;
        isRecording = true;
        while (isRecording & flag) {
            buffer = new byte[322];
            DatagramPacket receivePacket = new DatagramPacket(bBuffer, 322);
            serverSocket.receive(receivePacket);
            sourceLine.write(buffer, 0, buffer.length);
        }
    } catch (IOException e) {
        System.out.println("Receiving Error");
        System.exit(1);
    }
}
```

Výpis 8: Příjem dat a zapsání na výstup

8 Integrace do portálu her

Úkolem herního portálu by mělo být zajištění správy uživatelů, her a vztahy mezi nimi. Zaregistrovaný hráč by měl mít možnost se do portálu přihlásit, z nabídky si vybrat jednu z her a následně se k ní připojit. Úkolem herního portálu by mělo také být nahrazení odpojeného hráče z právě probíhající hry hráčem s umělou inteligencí. Portál by měl nabízet rozhraní pro začlenění nové hry.

Protože herní portál je bohužel stále ve fázi vývoje, není v mých silách tuto kapitolu dále rozšířit.

9 Umělá inteligence

Umělá inteligence je napodobením lidské inteligence vykonávané nějakým strojem či softwarem. Dnes je považováno také za samostatné odvětví počítačové vědy a výzkumu. Poprvé byl tento termín použit na Massachusettském institutu technologie roku 1956 [6].

V mém případě je podstatou umělé inteligence schopnost nahradit lidský faktor rozhodování. Munchkin je hra provázána rozhodováním a interakcí s ostatními hráči na mnoha místech. Hráči zvažují možnosti zahrát kartu, přijmout pomoc od hráčů nebo ji nabízet jiným, provádět akce a podobně.

Protože byla hra navržena do vrstev, je možné implementovat jednoduchou umělou inteligenci bez větších potíží. V klientské části aplikace je nahrazeno uživatelské rozhraní konzolí, do které se zadají údaje potřebné pro navázání spojení se serverem. Do konzole se následně jen vypisují provedené akce a informace o průběhu hry.

Akce se provádějí tak, aby byly nenáročné na interakci s ostatními hráči a výrazně nezasahovaly do vývoje hry, přičemž základním rozhodovacím faktorem je náhodné generování provedené akce. Z toho vyplývá, že hráč s umělou inteligencí by se měl začlenit do hry především z důvodu udržení hry v chodu, nebo pro spuštění hry dvou reálných hráčů a jedné umělé inteligence. Konkrétně tah umělé inteligence ve hře nyní vypadá následovně. Je-li na tahu umělá inteligence, provede se kontrola její karet v ruce. Nachází-li se v této kolekci karty, které lze v rámci fáze kola zahrát, vybere se z nich náhodně jedna a zahraje se.

Jak jsem již zmínil v předchozí kapitole, není možné aby hráč s umělou inteligencí nahradil odpojeného hráče od právě probíhající hry, protože není dokončen herní portál, který to bude mít na starost.

Jako jedno z možných rozšířeních mého zpracování by mohlo být naimplementování konkurenceschopné umělé inteligence, snažící se vyhrát.

10 Závěr

Hra na straně klienta je schopna komunikace s ostatními klienty připojenými ke hře v prostředí internetu. Hráči se můžou mezi sebou dorozumívat jak textovým, tak hlasovým chatem, což umožňuje vyjednávání a domlouvání spolupráce v rámci kola. O pravidla hry se stará herní logika implementovaná jako doménová logika softwaru. Podařilo se také naimplementovat hráče s jednoduchou inteligencí, vykonávajícího zásahy do hry, které jsou nenáročné na interakci. Bohužel se nepodařilo integrovat hru do portálu her, protože portál je stále ve stádiu vývoje. Jedním z kladených cílů bylo také algoritmizovat karty pomocí rozhraní, což se dle mého názoru povedlo jen z části, protože při podrobnější analýze významu karet bylo zjištěno, že rozhraní nebudou využita v takové míře, v jaké se předpokládalo.

Co znamenalo jmenovitě pro mne vypracování této bakalářské práce? Výzvu - jedním slovem definována pouť životním cyklem vývoje softwaru hry Munchkin v prostředí internetu. Na této cestě jsem se setkal s často využívanými technologiemi a programátorskými praktikami využívanými v pracovním světě programátora, mezi které patří vývoj rozsáhlých uživatelských rozhraní, zajištění komunikace skrze TCP a UDP protokol, plánování, návrh a analýza vyvíjeného softwaru. Ovšem utkal jsem se také s nevšedními nebo ne tak často řešenými problémy, jako získání hlasových dat, jejich zkompletování a následná zpětná interpretace, nebo implementace umělé inteligence.

K vývoji hry jsem nepoužil žádný herní engine, žádné nástroje pro usnadnění tvorby uživatelského rozhraní, ani žádné frameworky pro získání, zpracování a interpretaci hlasových dat při implementaci hlasového chatu a podobně. Ke všem řešeným problémům jsem se snažil přistupovat pouze využitím standardních Java knihoven, protože jedním z cílů kladených před započetím práce bylo omezení závislosti na zkonkretizovaných vývojových nástrojích.

Měl-li bych shrnout, co jsem si z cesty k „částečnému“ cíli odnesl, určitě bych zmínil mnohonásobné rozšíření mých obzorů v oblasti vývoje softwaru, co se týče možných praktik a přístupů k řešení problémů jako částí i jako celku, dále hluboké povědomí o všedních i nevšedních technologiích a zkušenosti s jejich implementací. Necíleným, ale pro mě velice přínosným důsledkem, bylo zlepšení orientace a schopnosti získávání potřebných znalostí z dokumentace Oracle.

Zmínil jsem se o „částečném“ cíli. K úplnému dokončení projektu by bylo potřeba mít k dispozici vyvinutý herní portál a přizpůsobit tento projekt jeho základním rysům. Jedním z nepřekonaných problémů, jenž nastal při vývoji, byl proces integrace hry Munchkin do technologie WebStart, protože nastaly komplikace se získáním podpisového certifikátu nezbytného pro spouštění skrze zmíněnou technologii. Dále by bylo potřeba testovat nasazený software dlouhodobě, nejlépe širokou veřejností a tak získávat údaje o nedostatcích a misinterpretacích a následně spravovat tyto nedostatky v rámci servisu.

Zde bych navázal na projekty, jež řeším na externích pracovištích, jmenovitě v práci. Protože pracuji právě na oddělení servisu, kde hledám příčiny problémů běhu softwaru, navrhuji jejich řešení a následně je implementuji, tak mám určité povědomí o důležitosti podpory nasazeného softwaru. Mimo to provádím také modifikace rozšiřující funkcionalitu, kde využívám například prvky uživatelského rozhraní, se kterými jsem se setkal při

realizaci toho projektu. O to větším přínosem pro mne bylo vypracování této bakalářské práce.

11 Reference

[1] RPG. *Zkratky.cz* [online]. 2008 [cit. 2014-04-27]. Dostupné z: http://www.zkratky.cz/RPG/16246

[2] *Word of Munchkin* [online]. 2001 [cit. 2014-04-27]. Dostupné z: http://www.worldofmunchkin.com/

[3] P2P Architecture. *Janalta Interactive Inc.* [online]. 2010-2014 [cit. 2014-04-27]. Dostupné z: http://www.techopedia.com/definition/454/peer-to-peer-architecture-p2p-architecture

[4] Peer-To-Peer Network. *The Government of the Hong Kong Special Administrative Region* [online]. 2008 [cit. 2014-04-27]. Dostupné z: http://www.infosec.gov.hk/english/technical/files/peer.pdf

[5] Client-Server Architecture. *Encyclopædia Britannica, Inc.* [online]. 2014 [cit. 2014-04-27]. Dostupné z: http://www.britannica.com/EBchecked/topic/1366374/client-server-architecture

[6] Artificial Intelligence. *Quin Street Inc* [online]. 2014 [cit. 2014-04-27]. Dostupné z: http://www.webopedia.com/TERM/A/artificial_intelligence.html

[7] Docs Oracle*Oracle* [online]. 1995-2014 [cit. 2014-04-27]. Dostupné z: http://docs.oracle.com

[8] Swing Components and Containers. *University of Minnesota Duluth* [online]. 2014 [cit. 2014-04-27]. Dostupné z: https://www.d.umn.edu/~gshute/java/swing/components.html

[9] Deserialization. *Stack exchange inc* [online]. 2012 [cit. 2014-04-27]. Dostupné z: http://stackoverflow.com/tags/deserialization/info

[10] Introduction to the AWT. *Java Word, Inc.* [online]. 1994-2014 [cit. 2014-04-27]. Dostupné z: http://www.javaworld.com/article/2077188/core-java/introduction-to-the-awt.html

[11] Java Archive Docs *Oracle* [online]. 2003-2010 [cit. 2014-04-27]. Dostupné z: http://download.java.net

[12] DARWIN, Ian F. *Java cookbook*. 2nd ed. Sebastopol, CA: O'Reilly, c2004, xxiv, 829 p. ISBN 05-960-0701-9.

[13] *Design patterns: elements of reusable object-oriented software*. Massachusetts: Addison-Wesley, c1995, xv, 395 s. ISBN 02-016-3361-2.

[14] Client/server and peer-to-peer models: basic concepts. *Department of Communications Engineering Tampere University of Technology* [online]. 2013 [cit. 2014-04-28]. Dostupné z: http://www.tut.fi/en

[15] N-tier Architecture and Tips. *CodeProject* [online]. 2012 [cit. 2014-04-28]. Dostupné z: http://www.codeproject.com

[16] Standard Widget Toolkit. *Eclipse* [online]. 2014 [cit. 2014-04-28]. Dostupné z: http://www.eclipse.org/swt/

A Příloha na CD

A.1 Karty

- obrázky (png)
- textová reprezentace karet použitá pro parsování (txt)
- analýza (xls)

A.2 Spustitelné soubory (jar)

- klientská aplikace (MunchkinClient)
- serverová aplikace (MunchkinServer)
- aplikace umělé inteligence (MunchkinAI)

A.3 Třídní diagramy (jpg)

Pro zobrazení třídních diagramů v plné velikosti jsou tyto diagramy také součástí CD.

- Třídní diagram dveří (door)
- Třídní diagram pokladů (treasure)
- Třídní diagram předmětů (item)

A.4 Zdrojové soubory

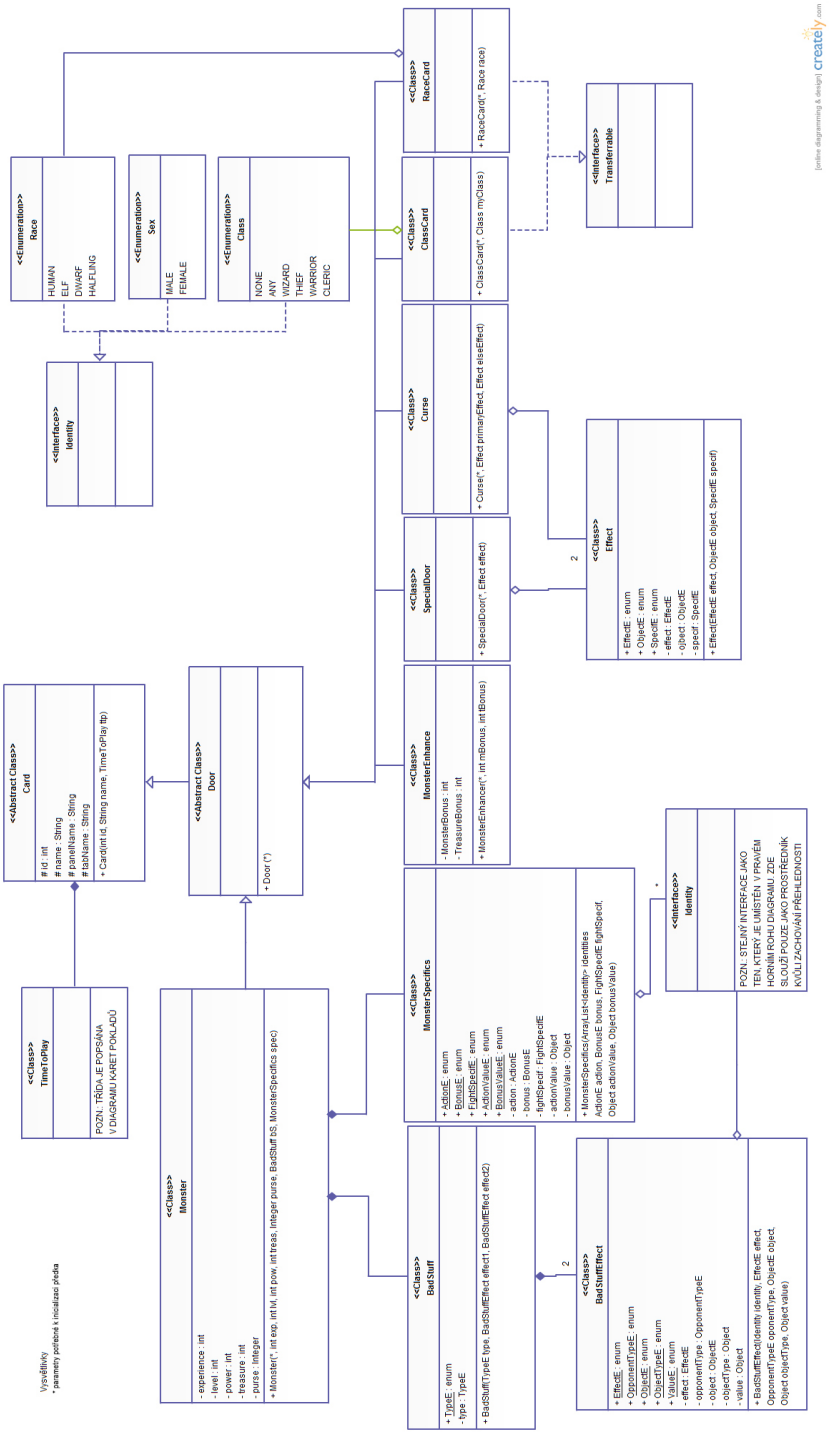
- klientská aplikace (MunchkinClient)
- serverová aplikace (MunchkinServer)
- aplikace umělé inteligence (MunchkinAI)

A.5 Uživatelská příručka (pdf)

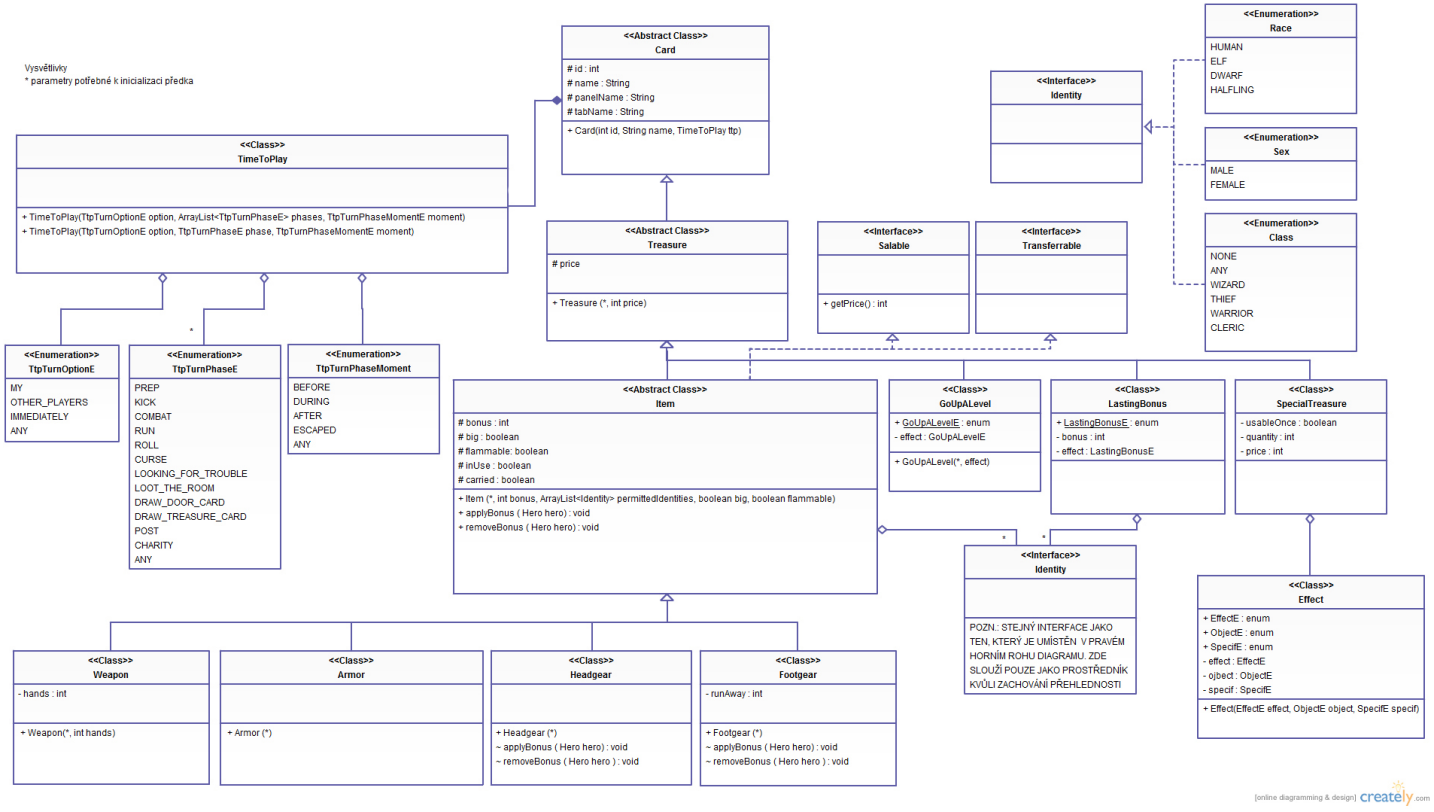
uživatelská příručka pro manipulaci a orientaci v grafickém uživatelském rozhraní klientské aplikace

- Munchkinova uživatelská příručka (4 strany)

B Třídní diagramy



Obrázek 5: Třídní diagram karet dveří (Door)



Obrázek 6: Třídní diagram karet pokladů (Treasure)